To appear in Proceedings of the 5th Symposium on Spatial Databases, SSD'97

Orthogonal Polygons as Bounding Structures in Filter-Refine Query Processing Strategies

Claudio Esperança[1] * and Hanan Samet[2] **

[1] COPPE, Prog. Eng. Sistemas
Universidade Federal do Rio de Janeiro
Cidade Universitária, C.T., Sala H319
Rio de Janeiro, RJ, 21945-970, Brazil
E-mail: esperanc@lcg.ufrj.br
[2] Computer Science Department and
Center for Automation Research and
Institute for Advanced Computer Science
University of Maryland at College Park
College Park, Maryland 20742
E-mail: hjs@umiacs.umd.edu

Abstract. The use of bounding structures in the form of orthogonal polygons (also known as rectilinear polygons) with a varying number of vertices in contrast with a minimum bounding rectangle (an orthogonal polygon with just 4 vertices in two dimensions) as an object approximation method is presented. Orthogonal polygons can be used to improve the performance of the refine step in the filter-refine query processing strategy employed in spatial databases. The orthogonal polygons are represented using the vertex representation implemented as a vertex list. The advantage of the vertex representation implemented as a vertex list is that it can be used to represent orthogonal polygons in arbitrary dimensions using just their vertices. This is in contrast to conventional methods such as the chain code which only work in two dimensions and cannot be extended to deal with higher dimensional data. Algorithms are given for varying the number of vertices used to represent the objects. It is shown that the use of non-trivial orthogonal polygons (i.e., with more than four vertices) is of benefit when a spatial index is used in the filter step for processing spatial queries such as point-in-object and windowing. If no spatial index is used, then all objects must be examined. In this case, many of the objects are small thereby not benefiting from the variation in the number of vertices that they have as the simple bounding box is adequate.

1 Introduction

The efficient processing of queries is an important issue in spatial databases [10]. This is facilitated, in part, through the use of spatial indexing

* The support of Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Projeto GEOTEC / PROTEM II is gratefully acknowledged.
** The support of the National Science Foundation under Grant IRI-92-16970 is gratefully acknowledged.

methods (e.g.,[22,23]) coupled with approximations. These techniques serve as the basis of a query processing strategy known as *filter-and-refine* [3]. In this case, the spatial indices serve to restrict the search to a subset of the data (termed the *filter* step) which is usually a subspace of the data. Once the subset has been defined, the queries are often executed with approximations of the underlying data as well as the query objects (termed the *refine* step). The rationale is to avoid excessive computation where the data is likely not to satisfy the query.

For example, determining the intersection point of two straight line segments is a surprisingly complex process due to issues of limited precision [15,17]. If we can determine that the segments do not intersect, then we can avoid the need to compute the intersection point thereby saving a considerable amount of effort. This can be done by approximating the two segments with some objects in which they are guaranteed to lie such as their enclosing rectangles, and then checking if the two enclosing objects overlap. This is a a relatively easy process especially if the rectangles are axes-aligned (i.e., their sides are parallel to the coordinate axes). This test can be made even more selective by ensuring that the rectangles fit the lines as closely as possible (termed a *minimum bounding rectangle (MBR)*).

We can generalize the line intersection problem to arbitrary curves in which case the selectivity of the refine step can be increased by loosening the requirement that the sides of the enclosing rectangles are parallel to the coordinate axes, thereby obtaining an enclosing object which is truly an MBR. Unfortunately, we are now confronted with our original problem of determining whether two arbitrary straight lines in space intersect since we must check if a pair of sides (one from each of the MBRs) intersect. Thus the only alternative is to loosen the restriction that the enclosing objects are rectangles and permit them to be orthogonal polygons[1] (also known as rectilinear polygons [5] having a varying number of vertices (i.e., more than four).

Interestingly, orthogonal polygons are not always the result of the approximation of the data. Instead, at times the original data is in the form of an orthogonal polygon due to the manner in which it was acquired (e.g., using raster conversion). For example, in a GIS application, orthogonal polygons can be used to indicate in what parts of a country a given crop is grown or the boundaries of lakes and water bodies.

These orthogonal polygons can be represented using the vertex representation [6,24]. The use of vertex representation as the representation of orthogonal polygons is the subject of this paper. The advantage of these methods is that they work for data of arbitrary dimensionality while still only recording the vertices of the orthogonal polygon. This is quite powerful as most higher-dimensional object representations make use of several of the

[1] In this paper we use the term *orthogonal polygon* in its most general sense to refer to data of arbitrary dimensionality.

lower dimensional components to represent the object. For example, in three dimensions, we may use the winged-edge representation [1,23] (also known as a BRep or a boundary model [16]) which represents the boundary of the object in terms of its vertices, edges, and faces.

In contrast, the vertex representation enables the representation of the object just using its vertices and the implicit assumption of orthogonal boundary elements. Notice that such implicit orthogonality assumptions about the boundary connectivity are also made when using chain codes [7] to represent two-dimensional data. However, it is impossible to generalize chain codes to higher dimensions as unlike the vertices (really edges) in two dimensions which are ordered in sequence of connectivity since each boundary element can be adjacent to just two elements, for $d > 2$ there is no such order associated with $(d-1)$-dimensional hyperplanes which are the boundary elements in d-dimensions.

The rest of this paper is organized as follows. Section 2 reviews the concept of a bounding structure and gives the key properties that make it useful in the processing of queries in spatial databases. Section 3 describes the use of the vertex representation of orthogonal polygons as bounding structures. This includes a definition of the vertex list implementation of the vertex representation. Section 4 discusses the actual process of dynamically varying the number of vertices in the bounding structures (termed *coarsening*) and gives an algorithm for doing so. The results of some experiments using the vertex representation as a bounding structure are given in Sect. 5 while concluding remarks are drawn in Sect. 6.

2 Properties of Bounding Structures

As mentioned in Sect. 1 in some situations it is advantageous to trade an exact geometric representation for an approximate one, provided it can be processed more efficiently. This is the key observation behind what we call the *bounding structure* technique. The most common type of bounding structure is the axes-aligned rectangle, mainly because of its simple geometry. Minimum bounding rectangles are used extensively in spatial databases, mainly in the design of spatial access methods. The R-tree [11] and its variations [2,25], for instance, rely heavily on the bounding properties of rectangles. Polygons, on the other hand, are in general too complex to be suitable as bounding structures, although they have been investigated and have been found promising in certain applications [4,13]. An alternative to the bounding structure as an object approximation is a decomposition of the space spanned by the object into a collection of cells (possibly, but not necessarily, disjoint). Such representations include the region quadtree [12,14] as well as the R^+-tree [25]. The variation in the number of cells as an approximation technique has been studied [8,18]. We do not discuss this approach further in this paper.

One of the main uses of bounding structures is to help in the evaluation of spatial predicates. Consider a spatial object S, where S may be a line,

a polygon, a collection of line segments or any other object that can be regarded as a set of points. We may use set notation to denote operations and predicates about spatial objects. For example, we may use predicate $S \cup R = \emptyset$ to say that spatial objects S and R do not overlap in space. Depending on the nature of S and R, the evaluation of such a predicate may vary in complexity from trivial (e.g., when both S and R are points) to very hard (e.g., when both S and R are sets of polygons). On the other hand, bounding structures are usually very simple geometric objects, so that evaluating common spatial predicates is never too costly.

One property of bounding structures that enables us to evaluate predicates on their corresponding bounded objects is the *subset property*. If $B(S)$ is a bounding structure for S, then S must be a subset of $B(S)$ (written $S \subseteq B(S)$). For example, if two objects have a non-empty intersection, then their bounding structures also intersect, i.e.:

$$S \cap R \neq \emptyset \Longrightarrow B(S) \cap B(R) \neq \emptyset.$$

An equivalent assertion is to say that if two bounding structures *do not* intersect, then their bounded objects cannot intersect either, i.e.:

$$B(S) \cap B(R) = \emptyset \Longrightarrow S \cap R = \emptyset.$$

Another frequent use of bounding structures is to help estimate distances between spatial objects. Let $\delta(a, b)$ denote the distance between two points a and b according to some metric. Then, the distance between two sets S and R, written $\delta(S, R)$, is defined as the minimum distance between two points belonging to S and R, respectively. Formally,

$$\delta(S, R) = \min_{s \in S, r \in R} \delta(s, r).$$

One could also consider the *maximum distance* (written $\Delta(S, R)$) as a way of defining the distance between two sets, i.e.,

$$\Delta(S, R) = \max_{s \in S, r \in R} \delta(s, r).$$

Bounding structures are of little use if the exact distance between two objects is to be computed. However, in many applications, distance computations are used in predicates to measure closeness between two objects, e.g., "$\delta(S, R) < 10$" or "$\delta(S, R) > 5$". In these cases, bounding structures can be used to give estimates which are sometimes sufficient to avoid computing the actual distance between the corresponding bounded objects. In particular, it is easy to see that the minimum and maximum distances between bounding structures provide bounds for the (minimum) distance between the bounded objects, i.e.,

$$\delta(B(S), B(R)) \leq \delta(S, R) \leq \Delta(B(S), B(R))$$

Depending on the nature of the bounding structure $B(S)$, it is possible to estimate a better (tighter) maximum bound for $\delta(S,R)$. As an example, consider a minimum bounding rectangle (MBR). If S is an object defined in $\mathcal{R}^d$, then its MBR has $2 \cdot d$ faces and, by construction, each face is guaranteed to contain at least one point of S; this is called the *MBR face property* [21]. If this property holds for bounding structure $B(S)$, then an upper bound for $\delta(S,R)$ can be established by computing the maximum distance between each pair of faces of $B(S)$ and $B(R)$ and taking the minimum of these. Let us call this measure $\Delta_\partial(B(S),B(R))$. Then, using the notation $\partial(B(S),i)$ to refer to each of the $2 \cdot d$ faces of $B(S)$, we can define

$$\Delta_\partial(B(R),B(S)) = \min_{i=1\ldots 2\cdot d, j=1\ldots 2\cdot d} \Delta(\partial(B(S),i),\partial(B(R),j))$$

Actually, we may think of $\partial(B(S),i)$ as any subset of $B(S)$ which is guaranteed to contain a point of S. Notice that $\Delta_\partial(B(R),B(S))$ is often[2] less than and can never be greater than $\Delta(B(R),B(S))$ (see Fig. 1).

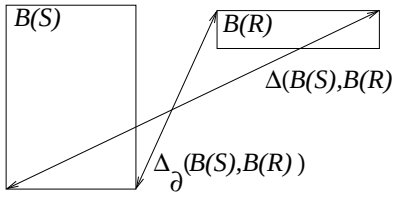


Fig. 1. Comparison between $\Delta(B(S),B(R))$ and $\Delta_\partial(B(S),B(R))$ for two sample objects.

We end this section by summarizing properties of a bounding structure $B(S)$ with respect to its corresponding bounded object S.

1. $S \subseteq B(S)$.
2. $B(S) \cap B(R) = \emptyset \Longrightarrow S \cap R = \emptyset$.
3. $\delta(B(S),B(R)) \leq \delta(S,R) \leq \Delta(B(S),B(R))$.
4. If we can extract from $B(S)$ (and $B(R)$) subsets $\partial(B(S),i)$ such that each of them contains at least one point of S, then $\delta(S,R) \leq \Delta_\partial(B(S),B(R)) \leq \Delta(B(S),B(R))$.

3 Orthogonal Polygons as Bounding Structures

An orthogonal polygon is a polygon whose sides are perpendicular to the coordinate axes. Axes-aligned rectangles are the most common instances of

[2] In fact, it can be shown that $\Delta_\partial(B(R),B(S))$ is always smaller than $\Delta(B(R),B(S))$, unless the boundary of $B(S)$ (and $B(R)$) is equal to $B(S)$ itself.

orthogonal polygons and the fact that they can be represented and processed efficiently has made them popular as bounding structures. Simple non-orthogonal polygons can be represented economically as a circular list of its endpoints and can be tuned to approximate a given region in two-dimensional space. However, certain key bounding structure operations are relatively hard to compute on polygons, e.g. testing whether two polygons intersect or computing the distance between them. Moreover, although polygons may easily be defined in multi-dimensional spaces, the effort to represent and process them in more than two dimensions makes them unattractive as bounding structures.

Recently, a new representation for orthogonal objects based on a data structure called the *vertex representation* has been presented [6,24]. This representation can be implemented in a manner that makes it possible to store an orthogonal polygon in any (integer) number of dimensions as a list of its vertices (termed a *vertex list*)[3]. Many common operations on orthogonal polygons represented by vertex lists may be efficiently computed by using algorithms based on the *sweeping plane* [20] approach. In particular, set operations such as those required for the processing of bounding structures have expected $O(d \cdot N)$ time complexity[6], where d is the number of dimensions and N is the total number of vertices in the vertex list(s). Our contribution is in the use of orthogonal polygons as bounding structures as well as the computation of morphological operations such as coarsening which is described in Sect. 4.

3.1 Vertex Representation

A polygon in two dimensions is usually represented as a circular list of its endpoints. This scheme cannot be extended to three or more dimensions since there is no obvious way in which vertices can be enumerated sequentially. The vertex list implementation solves his problem by using the order in which a hyperplane (e.g., a line in 2D or a plane in 3D) would encounter the vertices while sweeping the space from minus- to plus-infinity along one of the coordinate axes. Vertices that lie on the same hyperplane are similarly ordered by imagining a hyperplane of lower dimension sweeping that space. In addition to the coordinate values of its position in space, each vertex is accompanied by a scalar value termed the vertex *weight*[4]. Fig. 2 depicts an orthogonal polygon represented as a vertex list.

[3] Note that a polygon in three dimensions is called a *polyhedron* or, in general, a *polytope*. In this paper we will mostly deal with examples in two dimensions, and thus we opted to use the less precise term "polygon", although the described techniques can be applied to orthogonal polytopes of any number of dimensions

[4] Henceforth, we will use the term "vertex" to refer to weighted points, and the term "vertex list" to refer to a list of such weighted points. The term "vertex list" is not to be confused with the common way of representing two-dimensional polygons as a circular list of its endpoints.

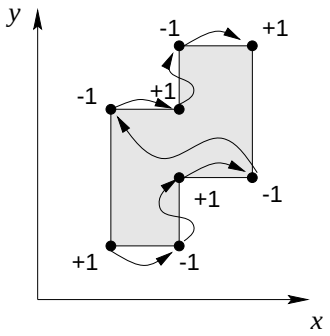


Fig. 2. An orthogonal polygon in 2D represented as a vertex list.

Intuitively, each vertex corresponds to the tip of an infinite cone. Each cone is equivalent to an unbounded object formed by the intersection of d (assuming d-dimensional data) orthogonal halfspaces that are parallel to the d coordinate axes passing through the vertex. The weights of the vertices have signs which serve to indicate whether the space spanned by their cones is included or excluded from the object being modeled. The space spanned by the union of the cones defines the object being modeled. Formally, the vertex representation is intended as a representation for orthogonal scalar fields. A scalar field is simply a function that maps points of the domain space to scalar values. An orthogonal scalar field is a scalar field where regions of the domain space which are mapped to the same value are delimited by faces orthogonal to the coordinate axes. The overall idea which enables the vertex representation to model orthogonal polygons is to think of these as scalar fields where points inside the polygon are mapped to 1, and points outside the polygon are mapped to 0. A vertex at a point $p = (p_1, p_2, \ldots p_d)$ and weight w has the effect of adding w to the mapping of all points $q = (q_1, q_2, \ldots q_d)$ such that $q_i \geq p_i$ for all $i = 1 \ldots d$ (see Fig. 3a). As a consequence, one can tell if a point q is inside or outside the polygon by adding the weights of all vertices that contribute to its mapping, i.e., a vertex at point p contributes to the mapping of q if and only if $p_i < q_i$ for all $i = 1 \ldots d$ (see Fig. 3b).

3.2 Fitting an Orthogonal Polygon around a Spatial Object

The general procedure for creating an orthogonal polygon $B(S)$ that can serve as a bounding structure for a spatial object S consists of:

1. Partitioning S into smaller objects $s_1, s_2 \ldots s_k$.
2. Finding the minimum bounding rectangle $MBR(s_i)$ for each sub-object s_i.
3. Computing the union of all MBRs: $\bigcup_{i=1}^{k} MBR(s_i)$.

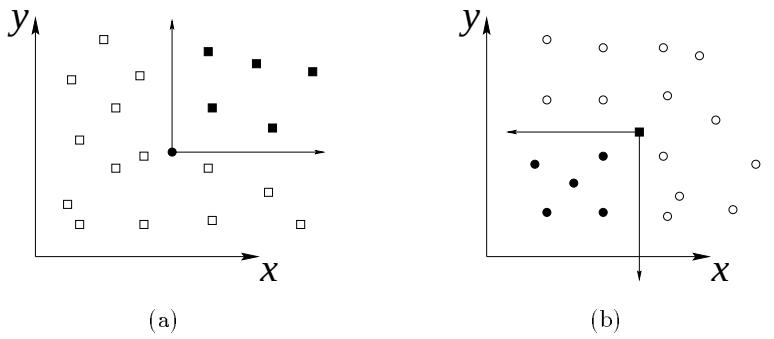


Fig. 3. (a) A vertex (black dot) and some of the points influenced by it (black squares). (b) The point (black square) is only influenced by the vertices shown as black dots.

The problem of partitioning an object into smaller objects is frequently trivial (e.g., a set of line segments can be partitioned by taking each line segment separately) or may require some additional processing. For example, we might use triangulation [19] to split a (simple) polygon into triangles. An MBR for each triangle can be computed easily yielding four vertices which can then be stored in a list. To compute the union of all MBR's, we use the algorithm for performing set operations on vertex list implementations of vertex representations as described in [6]. This is illustrated in Fig. 4.

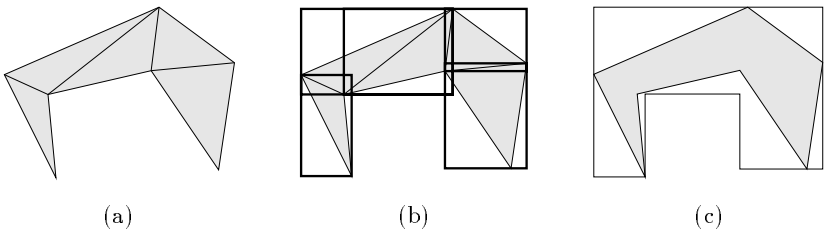


Fig. 4. (a) Polygon is split into triangles, (b) an MBR is fitted around each triangle, (c) taking their union produces an orthogonal bounding polygon.

In some cases, the boundary of a spatial object is amenable to trivial partitioning, while the object itself is not. A general polyhedron, for instance, is delimited by polygonal faces and these can be partitioned with little difficulty by using triangulation as described above. However, the partitioning of its interior, say into tetrahedrons, is rather complicated. In such cases, we may compute an orthogonal bounding polygon for the object by performing

a morphological operation known as *closing*[5] on its boundary. The operation of *closing* for rectangular structuring elements can be computed efficiently on orthogonal objects represented by a set of vertices and stored as a vertex list [6]. Loosely speaking, *closing* has the property of filling gaps and holes in polygons. A more precise description of the procedure is as follows:

1. Compute $MBR(S)$, the minimum bounding rectangle of object S.
2. Partition the *boundary* of S into components $s_1, s_2 \ldots s_k$.
3. Find the minimum bounding rectangle $MBR(s_i)$ for each sub-object s_i.
4. Compute the union of the MBRs of all subobjects, i.e., $\bigcup_{i=1}^{k} MBR(s_i)$.
5. Perform a *closing* operation on the orthogonal polygon obtained in (4) using $MBR(S)$ as the structuring element.

This procedure is illustrated in Fig. 5 for a simple polygon in two dimensions. A side effect of this method is that it always produces orthogonally convex polygons,[6] which might be undesirable for non-convex objects.

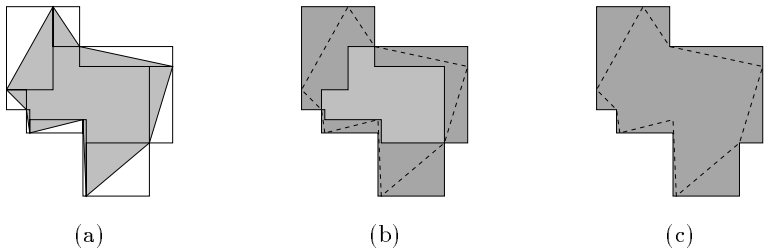


Fig. 5. (a) An MBR for each edge of the polygon is computed. (b) The union of these may produce an orthogonal polygon with holes. (c) A *closing* operation fills eventual holes and gaps.

It is not difficult to see that the orthogonal polygons obtained through the use of the above methods satisfy the bounding structure properties outlined in Section 2. Nevertheless, if $B(S)$ is such a polygon, then property 4 requires us to define a method for obtaining suitable subsets $\partial(B(S), i)$. We notice that both methods for building orthogonal polygons use unions of MBRs as starting points and, although the faces of the MBRs are guaranteed to contain at least one point of the bounded object, the faces of the resulting

[5] Due to space limitations, we cannot provide here background information on morphological operations. We encourage readers who might be unfamiliar with the subject to refer to any good text on image processing (e.g., [9]).

[6] A polygon in two dimensions is orthogonally convex if its intersection with any horizontal or vertical line has at most one connected component [19]. This definition can be extended to any number of dimensions by replacing "line" with "hyperplane" and "horizontal/vertical" with "normal to a coordinate axis".

orthogonal polygon do not necessarily do so (see Fig. 6). However, some faces of such polygons do have that property, namely those that constitute local minima or maxima. A face f is said to constitute a local minimum or maximum if all adjacent faces lie to the same side of the plane defined by f. Such faces can easily be extracted from a vertex list implementation of a vertex representation by means of a plane-sweep algorithm [6]. Note that local minimum and maximum faces can then be used as subsets $\partial(B(S), i)$ in the context of property 4 as explained in Sect. 2.

Fig. 6. Although faces of minimum bounding rectangles are guaranteed to contain at least one point of the bounded objects (a), the faces of the orthogonal polygon resulting from their union (b), unless they are local maxima or minima (e.g., faces drawn with thin lines), do not necessarily preserve that property (e.g., faces drawn with thick lines).

4 Coarsening Orthogonal Polygons

The primary objective of bounding structures is to provide a good approximation of the bounded object while taking as little space as possible. Thus, when we consider using an orthogonal polygon for this purpose, it is essential to be able to achieve an optimum tradeoff between tightness and space. It is possible to obtain a vertex representation which is as close to a given spatial object as desired. All that is necessary is to partition that object into a suitable number of small fragments and apply the procedure outlined in Section 3.2. However, the vertex representation produced in this way may be too complex (i.e., it may have too many vertices). Therefore, some procedure must be devised which will produce less detailed (coarser) versions of orthogonal bounding polygons while preserving, as much as possible, their usefulness as a bounding structure. In this section, we describe two approaches to solve this problem.

4.1 The "Moving Faces" Approach

This approach derives from the observation that, in general, if a face which is perpendicular to one of the coordinate axes is displaced along that axis, the total number of vertices of the representation will never increase. By choosing an appropriate displacement, it is possible to guarantee that the resulting polygon will not only be coarser (i.e., have less vertices) than the original, but will also preserve the bounding properties outlined in Section 2.

Given an orthogonal polygon in d dimensions represented by a vertex list, it is possible to sweep a plane along the d^{th} coordinate axis and extract the faces perpendicular to that axis by a simple sequential scan of the list. These faces can be divided into two groups, which we term the *positive* and *negative* faces. A positive face corresponds to a region where the sweeping plane *enters* the polygon in its movement from minus- to plus-infinity along the d^{th} coordinate axis, whereas a negative face corresponds to a region where the sweeping plane *leaves* the polygon (see Fig. 7a). Coarsening is achieved

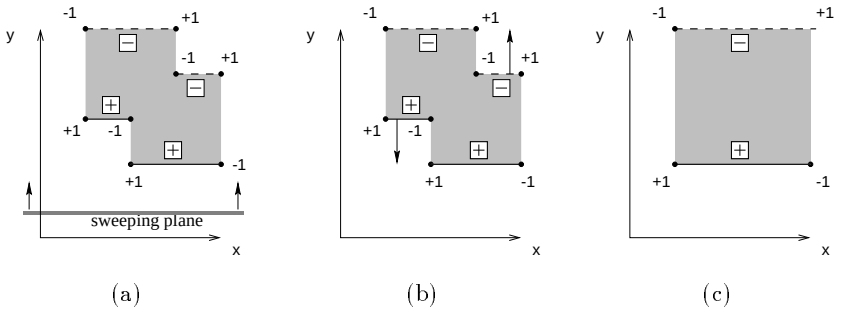


Fig. 7. 2D example of coarsening with the "moving faces" approach. Positive and negative faces are identified (a) and moved away from the polygon until they meet another face of the same sign (b), resulting in a coarser polygon (c).

by moving a positive face until it touches another positive face or, similarly, moving a negative face until it touches another negative face. As we must maintain the bounding properties of the polygon, faces must be moved in directions that will result in polygon expansion. This means that positive faces are moved in the opposite direction of the coordinate axis and negative faces are moved in the same direction of the coordinate axis (see Fig. 7b). Once faces are moved in this fashion, some vertices of the moved face will cancel out vertices of the face to which it is now adjacent, thereby resulting in a coarsened polygon (Fig. 7c).

Notice that since no faces are actually created in the process, faces which were guaranteed to contain a point of the bounded object still do so in the coarsened polygon, thus preserving property 4 as described in Section 2.

A full implementation of the "moving faces" approach requires that the process be repeated for faces perpendicular to all coordinate axes. This can be achieved by reordering the list so that a plane swept along each axis may find the corresponding faces. Also, the choice of which faces to move must take in consideration the size of each candidate face and the amount of displacement required. A rule of thumb is to choose the face displacements which will result in the least enlargement of the original polygon.

4.2 The "Rectangle Pairs" Approach

The moving faces approach will fail to produce a coarsened polygon in some cases. For example, some polygons with holes or having more than one connected component (see Fig. 8) may not contain suitable positive or negative faces. In such cases, a more general approach is needed.

Fig. 8. Some polygons with holes (a) or having more than one connected components (b) cannot be coarsened with the "moving faces" approach.

In the "rectangle pairs" approach, the original polygon is first divided into a set of rectangles (see Fig. 9a). ¿From these, a few pairs are chosen and, for each pair, a minimum bounding rectangle (MBR) is constructed (Fig. 9b). The union of the original polygon and MBRs will result in a coarsened polygon (Fig. 9c). The overall idea is that each selected pair of rectangles should contain a number of vertices of the original polygon and, by replacing them with a minimum bounding rectangle, some of these vertices will be eliminated from the representation. A quick calculation shows that, since in d dimensions a rectangle has 2^d vertices, the substitution of two rectangles r_a, r_b by their minimum bounding rectangle $MBR(r_a, r_b)$ may eliminate up to 2^d vertices from the representation.

In an actual implementation of this algorithm, we must address the following points:

1. How to divide an orthogonal polygon into a set of rectangles? This problem can be solved with Shechtman's algorithm for splitting a polygon in maximal horizontal rectangles in two dimensions [24]. This algorithm was

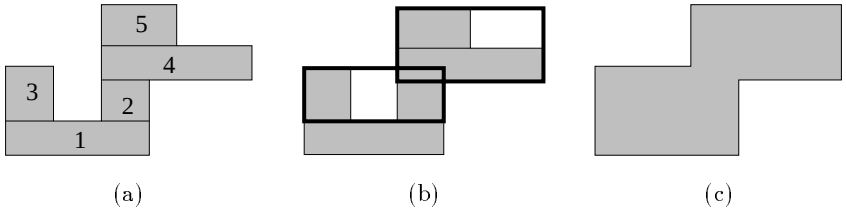


Fig. 9. Coarsening with the "rectangle pair" approach. Polygon is divided into rectangles (a), some pairs are chosen (2-3 and 4-5) and a MBR for each pair is computed (b). Coarsened polygon is union of MBRs and original polygon (c).

extended in [6] to deal with polygons in arbitrary dimensions. Fig. 9a is an example of such a subdivision.

2. Which rectangle pairs to choose? Ideally, rectangles which are close to each other should be considered first. Another consideration is the total amount of "wasted space" introduced when substituting a pair of rectangles by its MBR. The algorithm for dividing polygons into rectangles returns those rectangles in the same order in which the sweeping plane finds them. Although this order does not guarantee that two consecutive rectangles are the closest pair to each other, it does provide a certain amount of clustering. Thus, the selection of pairs of rectangles may consider consecutive rectangles and reject those which are too distant from each other or those whose pair-wise MBR is too large compared with the size of the rectangles.

4.3 The Complete Algorithm

We have implemented a complete coarsening algorithm using both approaches just described. The algorithm takes as input a d-dimensional orthogonal polygon $B(S)$ with n vertices and the desired maximum number of vertices k. Its output is an orthogonal polygon containing no more than k vertices that can be used as a bounding structure for S satisfying the properties outlined in Section 2.

The algorithm consists of a heuristic method for combining the strengths of both approaches in a controlled way. The ultimate goal is to obtain a bounding structure of limited complexity which nevertheless approximates the bounded object as closely as possible. A reasonable strategy involves estimating how much the polygon will grow as each modification prescribed by either approach is applied. In the "moving faces" approach, selecting a face f and displacing it by i units can be seen as computing the union of the original polygon with a "plug" polygon given by sweeping f perpendicularly by i units. Similarly, in the "rectangle pairs" approach, if the MBR of a pair of rectangles ($r1$,$r2$) is used in the coarsening process, then we may estimate

that the original polygon will grow by an amount represented by the regions of $MBR(r1, r2)$ not already occupied by $r1$ or $r2$. In other words, we can imagine that we applied to the original object a "plug" polygon corresponding to $MBR(r1, r2) \setminus (r1 \cup r2)$[7]. Overall, we can consider each plug shape as a possible modification that will coarsen the original polygon. The amount of growth due to this modification can be estimated as the integral (e.g. area in 2D or volume in 3D) of the plug. This enables us to eliminate plug polygons which are too large.

The complete algorithm is summarized below.

1. If the length of list $B(S)$ is less or equal to k, then return $B(S)$.
2. Initialize array $Plug$ which will hold plug polygons.
3. Repeat the following steps for all coordinate axes:
   (a) Use the "moving faces" approach to compute faces which can be displaced to coarsen $B(S)$. For each candidate face f, compute also the corresponding displacement i. Create a plug polygon by sweeping f by i units and add it to $Plug$.
   (b) Use the "rectangle pairs" approach. Perform a rectangle partition of the polygon. Take every two consecutive rectangles ($r1$,$r2$) as returned by the rectangle partitioning algorithm and compute their pair-wise MBR. Compute each plug polygon given by $MBR(r1, r2) \setminus (r1 \cup r2)$ and add it to $Plug$.
   (c) Reorder the vertices of vertex list $B(S)$ so that plane sweeping may occur along another coordinate axis.
4. Compute the integral of each polygon in $Plug$ and sort $Plug$ in increasing order of integrals.
5. If $Plug$ has m polygons at this point, then discard the $\lfloor m/2 \rfloor$ largest polygons. This heuristic aims to consider only "small" plug polygons. Applying "big" plug polygons might lead to a faster coarsening, at the expense of achieving worse approximations. Note that up to $m - 1$ plug polygons could be discarded, but at least one plug polygon must be considered so that the coarsening process terminates.
6. Scan $Plug$ sequentially and apply each plug polygon to $B(S)$ (i.e, compute the union of $B(S)$ and the plug) until the resulting polygon has k vertices or less or $Plug$ is exhausted. In the latter case, return to step 2.

Step 6 deserves a more detailed explanation. Since $B(S)$ and plug polygons are represented by of vertex lists, computing their union takes time proportional to the sum of their (list) lengths [6]. Hence, if $B(S)$ is a fairly long list, then repeatedly computing its union with each plug polygon would be too costly. Fortunately, there is no need to compute the union for each plug polygon, only to count how many vertices the resulting polygon will have. In practice, one may use a data structure, say $Match$, which is suitable for storing and searching points in d dimensions (e.g., a k-d-tree [23]).

[7] The symbol "\" is used here to denote set difference.

Initially, *Match* is initialized with all vertices of $B(S)$. Then, as each plug polygon is considered, *Match* is searched for vertices at identical positions. When matching vertices are found and these have opposite signs, the length of the result would decrease by 1; if not, the length of the result increases by 1. Once all plug polygons are processed in this fashion, the resulting coarsened $B(S)$ can be built by collecting all vertices in *Match*.

The time complexity analysis of this algorithm is rather involved and is not given here (see [6]). However, we state without proof that the time complexity of the coarsening algorithm can be estimated at $O(2^d \cdot N \cdot \log^2 N)$ on average, where N is the number of vertices in the original polygon. The $2^d \cdot N$ factor is the average number of vertices in all plug polygons generated in step 3. One $\log N$ factor comes from the search used in *Match* for identical vertices. The remaining $\log N$ factor arises from step 5 which causes the algorithm to be performed $\log N$ times.

It must be observed that the approach described above for obtaining coarse approximations of complex objects is not too attractive if the desired approximation contains much fewer vertices than the original object. For instance, if we are trying to obtain an MBR for a complex object, a straightforward algorithm that computes the limits of the object in each dimension is much more efficient. Thus, in these cases, an alternative approach would consist of starting with an MBR and, by means of successive "erosion" steps, produce a finer approximation until the desired level of approximation is obtained. Such an algorithm is currently being investigated.

5 Experiments

In this section we empirically evaluate both the proposed coarsening algorithm and the usefulness of orthogonal polygons as bounding structures. Whereas bounding structures are used for many purposes in several application areas, we focus on a typical application of bounding structures in the context of a spatial database system.

The experiments were conducted on data extracted from a digital map depicting forest coverage types in Canada. A raster image of this map was provided by Canada's National Atlas Information Service[8] and is shown in Fig. 10. The data consists of a collection of orthogonal polygons delimiting the homogeneous 4-connected regions of the raster image. The algorithm for converting raster images into a set of orthogonal polygons stored as vertex lists is the one described in [6]. A total of 403 polygons ranging from 4 to 1408 vertices (average 50.8) were found.

[8] The URL `http://elllesmere.ccm.emr.ca/naismap/naismap.html` locates this data in the World Wide Web

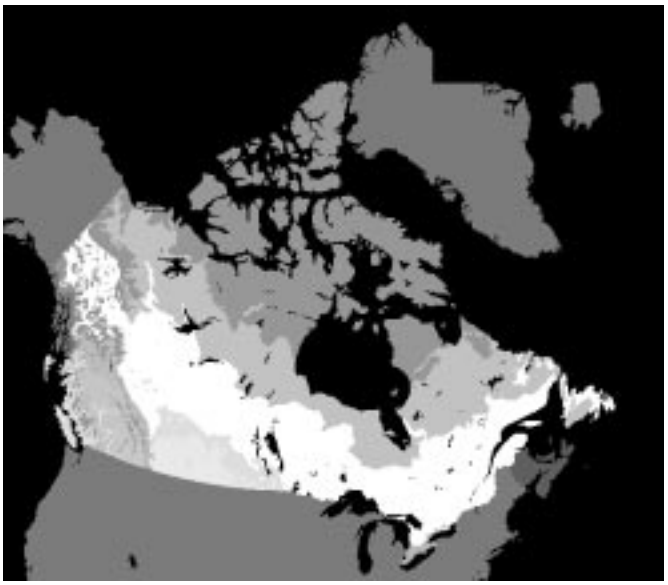


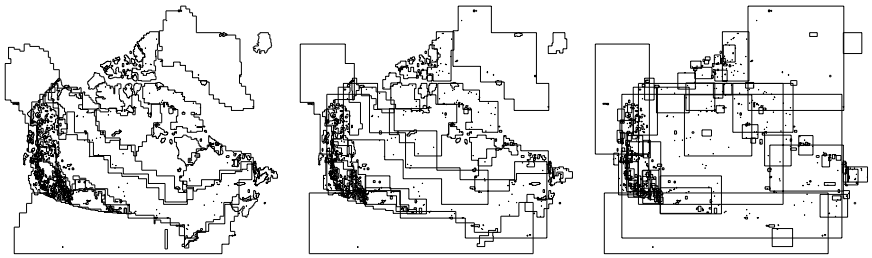
Fig. 10. Forest coverage by type in Canada

5.1 Coarsening experiments

A series of experiments were conducted to determine the general behavior of the proposed algorithm in coarsening the polygons of the sample data set. All polygons were first coarsened to no more than 256 vertices. This limit was repeatedly halved until the coarsened polygons contained exactly 4 vertices, which is a lower bound for polygons in two dimensions. Fig. 11 shows the result of coarsening the polygons to no more than 64, 16 and 4 vertices, respectively.

The graph in Fig. 12 shows the time taken by the coarsening algorithm to produce the 4 vertex MBR of the polygons of different sizes. Although the final 4 vertex MBRs could have been obtained just as easily through the standard procedure of finding the minimum and maximum values of the x and y coordinate values, our goal was to determine how the complexity (i.e., number of vertices) of the original polygon impacts the speed of coarsening. As can be observed from Fig. 12, the graph curve is somewhat irregular due to the small number of polygons of high complexity. Nevertheless, the time complexity of the algorithm seems to be in accordance with the estimate presented in Section 4.3.

We were also interested in evaluating how coarsening affects the area of an orthogonal polygon. The area of a coarsened polygon, when compared with that of the original polygon, provides a measure of how tightly one shape bounds the other. With this in mind, for each coarsening level we computed



(a) (b) (c)

Fig. 11. Map of Canada's forest coverage by type, where each orthogonal polygon was coarsened to no more than 64 (a), 16 (b) and 4 vertices (c).

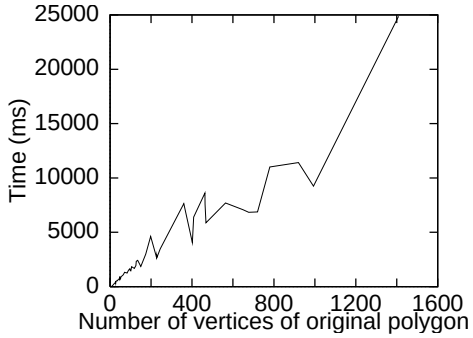
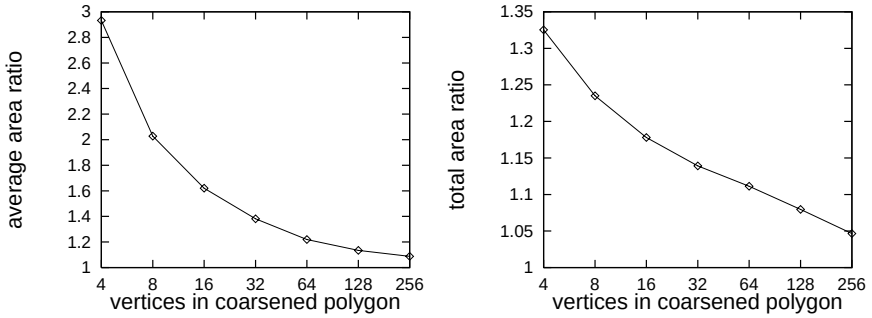


Fig. 12. Plot of times taken by algorithm *coarsen* to reduce to 4 the number of vertices of orthogonal polygons.

the result of taking the sum of the areas of all coarsened orthogonal polygons and dividing that value by the sum of the areas of the original polygons (i.e., their ratio). The plots of these ratios as a function of the coarsening level are shown in Fig. 13a. Note that in this evaluation, total areas for the whole data set were considered, which leaves room for inordinately large polygons to influence the result. This effect can be reduced if we compare relative areas on a per-polygon basis. The relative area consists of the fraction obtained by dividing the area of a coarsened polygon by that of the corresponding original polygon. In Fig. 13b we plot average relative areas.

5.2 Adaptive coarsening

In a spatial database environment, a data set such as the one we have chosen for our experiments could be accessed on the basis of several selection



(a) (b)

Fig. 13. Plot of ratio between areas covered by orthogonal polygons after and before coarsening. (a) Refers to the ratio between the total area of all coarsened polygons and the total area of all original polygons. (b) Shows the average area ratios between each coarsened and non-coarsened polygon.

criteria. A very common operation in spatial databases is the retrieval of objects that satisfy a condition expressed in terms of spatial proximity. The processing of such queries frequently relies on heuristics employing bounding structures to prune out objects which cannot possibly satisfy the condition. The effectiveness this pruning process depends on two factors:

1. How closely the bounding structure approximates the bounded object.
2. How much extra work must be done to determine if the bounded object satisfies the condition in case the test against the bounding structure is not conclusive.

This suggests that in order to compute a bounding structure of optimal complexity for a given object one has to take into consideration not only the object's complexity but also its extent (i.e., size).

Consider a simplified scenario where a collection of objects is queried in order to select the ones which intersect a single point p (termed the *point intersection* query). We will assume that the position of p is uniformly distributed over a rectangular region A (e.g., the limits of the original map image). Thus, we can estimate the following costs incurred in deciding whether a sample object S intersects p:

1. Determine if $B(S)$ intersects p. This can be done at a cost that is linearly dependent on the complexity (number of vertices) of $B(S)$. If we stipulate that constant c is an estimate of the per-vertex cost of an intersection computation, then the total cost of this first step is:

$$C_1 = c \times length(B(S)).$$

2. Test S against p. In our case, since S is also represented by a vertex list, the cost of such a test is given by $c \times length(S)$. However, this step only takes place if $B(S)$ intersects p. The probability of this event is proportional to area of $B(S)$. In the limit, that is, when $B(S)$ is equal to A, the probability is 1. Therefore, we may estimate that the cost due to this step is

$$C_2 = c \times length(S) \times \frac{area(B(S))}{area(A)}.$$

Cost estimates C_1 and C_2 were used to perform an adaptive coarsening of the polygons in our sample data set. Starting with a very fine approximation of each polygon, this was progressively coarsened until the value of $C_1 + C_2$ ceased to decrease. Since this process only requires that successive cost estimates be compared, the actual value of constant c is of no importance. Notice that if the successive values of $C_1 + C_2$ do not decrease at all, the first (finest) approximation is used. If they decrease monotonically, the coarsest approximation (i.e. the MBR) is used. One could also argue that many inflection points in this sequence of values could exist thereby leading to a false stopping point; however, this did not occur in our experiments (see Section 5).

One last consideration must be made concerning the sampling area represented by A. The assumption that A spans the space covered by all of the map is consistent with a query processing strategy based on sequential search. A more realistic scenario, however, must allow for the use of spatial indices. In other words, if a spatial index is used to filter out values which cannot satisfy the query, then a polygon will be tested only if the index itself was incapable of giving a positive diagnostic. For example, if polygons are indexed by means of an R-tree [11], then any given polygon will be tested only if p falls within its MBR. We can easily modify our adaptive coarsening procedure to take spatial indices into consideration by adjusting the value of A to that of $MBR(S)$.

In our experiments, adaptive coarsening produced the following average number of vertices for $B(S)$:

Average number of vertices	
Sequential search	4.1649
Spatial Index	4.9123

When using a sequential search strategy to test the entire set of polygons against the query point, we find that the average number of vertices at the time the adaptive coarsening process halted was very close to 4 (i.e., 4.1649) which means that the resulting bounding structure was a rectangle (i.e., an orthogonal polygon with 4 vertices). This implies that it is not worth our time to use a bounding structure that has a varying number of vertices. Instead, we can use an MBR. On the other hand when using a spatial index to decide which polygons should be tested against the query point, a tighter

bounding structure does lead to better results as can be seen by the fact that the average number of vertices when the adaptive coarsening process halted was close to 5 (i.e., 4.9123). These results are consistent with our intuition in the sense that executing the point intersection query by testing every polygon against the point (i.e., using sequential search) requires testing $B(S)$ for every value of S in the data set, and thus this test should be as fast as possible. In contrast, a spatial index already provides a good way of pruning the search based on spatial proximity, so that fewer objects are tested. Thus, in order to be reduce the set of candidate objects from consideration, it is necessary to use tighter, i.e., more elaborate bounding structures. Loosely speaking, we may think that, in this case, the spatial index provides a rough approximation for the spatial distribution of the objects, while tighter bounding structures are used in a second stage to prune the search even further before testing the actual objects.

5.3 Window query experiments

A series of window query experiments were conducted in order to evaluate the usefulness of adaptive bounding structures in a more realistic spatial database environment. This type of query consists in selecting all objects which intersect a given rectangle, or "window". This query is a two-dimensional generalization of the point intersection query discussed in Section 5.2. Experiments were made for window sizes ranging from 1% to 50% of the total map area using both sequential search and spatial index-based strategies. The same experiments were also conducted using MBRs (as opposed to adaptive orthogonal polygons) and using no bounding structures whatsoever. The experiments involving spatial indices used a PMR-quadtree [23] with a maximum depth of 10, and a splitting threshold value of 16. Fig. 14a shows the total query evaluation times for the experiments using sequential search, and Fig. 14b shows the equivalent results obtained with a spatial index based strategy.

One conclusion that can be drawn from these results is that the more flexible bounding structures derived from our adaptive approach lead to better query performance in all cases. The difference is more evident in the tests using a spatial index, but this was expected since the adaptive scheme only prescribed bounding structures significantly more complex than MBRs when indices were used. Further evidence that adaptive bounding structures are more beneficial when used in conjunction with spatial indices is shown in Fig. 15 which plots the average number of times in which bounding structures failed to determine if the object satisfied the query.

Overall, although the empirical tests do not demonstrate a dramatic improvement when adaptive bounding structures are used, they suggest that bounding structures of variable complexity might lead to significantly enhanced processing of some queries, e.g. window selection queries for large windows (see Fig. 14).

(b)

Fig. 14. Comparison of running times for different bounding structure configurations, using the sequential search (a) and the spatial index-based query evaluation plans (b).

6 Concluding remarks

The use of bounding structures in the form of orthogonal polygons with a varying number of vertices in contrast with a minimum bounding rectangle (an orthogonal polygon with just 4 vertices in two dimensions) as an object approximation method is presented. Orthogonal polygons can be used to improve the performance of the refine step in the filter-refine query processing strategy employed in spatial databases. The orthogonal polygons are represented using the vertex representation implemented as vertex lists. The advantage of the vertex representation is that it can be used to represent orthogonal polygons in arbitrary dimension using just their vertices. This is in contrast to conventional methods such as the chain code which only work in two dimensions and cannot be extended to deal with higher dimensional data.

Algorithms were given for varying the number of vertices used to represent the objects. We showed that use of non-trivial orthogonal polygons (i.e., with more than four vertices) is of benefit when a spatial index is used in the filter

(b)

Fig. 15. Relative number of times bounding structure tests fail to give a positive answer; (a) rates for the sequential search plan, and (b) rates for the index-based plan.

step for processing spatial queries such as point-in-object and windowing. If no spatial index is used, then all objects must be examined. In this case, many of the objects are small thereby not benefiting from the variation in the number of vertices that they have as the simple bounding box is adequate.

Future work includes a thorough comparison between the adaptive bounding structures presented in this paper and other fixed-size bounding structures proposed, e.g., in [3]. We also plan to incorporate vertex representations implemented as vertex lists in real spatial database query processor and building query processing strategies around its use. These strategies would be embedded in a query optimizer for queries containing conditions that involve values of both spatial and non-spatial attributes.

References

1. B. G. Baumgart. Geometric modeling for computer vision. Stanford Artificial Intelligence Laboratory Memo AIM-249 STAN-CS-74-463, Stanford University, Stanford, CA, October 1974.

2. N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The R^*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the SIGMOD Conference*, pages 322–331, Atlantic City, NJ, June 1990.
3. T. Brinkhoff, H. P. Kriegel, R. Schneider, and B. Seeger. Multi-step processing of spatial joins. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 197–208, Minneapolis, MN, June 1994.
4. A. Brodsky, C. Lassez, J. Lassez, and M. J. Maher. Separability of polyhedra for optimal filtering of spatial and constraint data. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 54–65, San Jose, CA, May 1995.
5. H. Edelsbrunner, J. O'Rourke, and E. Welzl. Stationing guards in rectilinear art galleries. *Computer Vision, Graphics, and Image Processing*, 27(2):167–176, August 1984.
6. C. Esperança. *Orthogonal Objects and their Application in Spatial Databases.* PhD thesis, University of Maryland, December 1995. (also available as Technical Report TR-3566, University of Maryland, College Park, MD).
7. H. Freeman. Computer processing of line-drawing images. *ACM Computing Surveys*, 6(1):57–97, March 1974.
8. V. Gaede. Optimal redundancy in spatial database systems. In M. J. Egenhofer and J. R. Herring, editors, *Advances in Spatial Databases — Fourth International Symposium, SSD'95*, pages 96–116, Portland, ME, August 1995. (also Springer Verlag Lecture Notes in Computer Science 951).
9. R. C. Gonzalez and R. E. Woods. *Digital Image Processing.* Addison-Wesley, Reading, MA, June 1992.
10. R. H. Güting. An introduction to spatial database systems. *Special Issue on Spatial Database Systems of the VLDB Journal*, 3(4), October 1994.
11. A. Guttman. R–trees: a dynamic index structure for spatial searching. In *Proceedings of the SIGMOD Conference*, pages 47–57, Boston, MA, June 1984.
12. G. M. Hunter. *Efficient computation and data structures for graphics.* PhD thesis, Princeton University, Princeton, NJ, 1978.
13. H. V. Jagadish. Spatial search with polyhedra. In *Proceedings of the Sixth IEEE International Conference on Data Engineering*, pages 311–319, Los Angeles, February 1990.
14. A. Klinger. Patterns and search statistics. In J. S. Rustagi, editor, *Optimizing Methods in Statistics*, pages 303–337. Academic Press, New York, 1971.
15. G. D. Knott and E. D. Jou. A program to determine whether two line segments intersect. Department of Computer Science TR–1884, University of Maryland, College Park, July 1987.
16. M. Mäntylä. *An Introduction to solid Modeling.* Computer Science Press, Rockville, MD, 1987.
17. J. Nievergelt and P. Schorn. Das ratsel der verzopften geraden. *Informatik-Spektrum*, 11:163–165, 1988. (in German).
18. J. A. Orenstein. Redundancy in spatial databases. In *Proceedings of the SIGMOD Conference*, pages 294–305, Portland, OR, June 1989.
19. J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, 1994.
20. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer–Verlag, New York, 1985.

21. N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the ACM SIGMOD Conference*, pages 71–79, San Jose, CA, May 1995.
22. H. Samet. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, Reading, MA, 1990.
23. H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.
24. J. Shechtman. Processamento geométrico de máscaras VLSI. Master's thesis, Eng. Elétrica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, April 1991.
25. M. Stonebraker, T. Sellis, and E. Hanson. An analysis of rule indexing implementations in data base systems. In *Proceedings of the First International Conference on Expert Database Systems*, pages 353–364, Charleston, SC, April 1986.